****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **June 29, 2025** | **Kane Miller** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

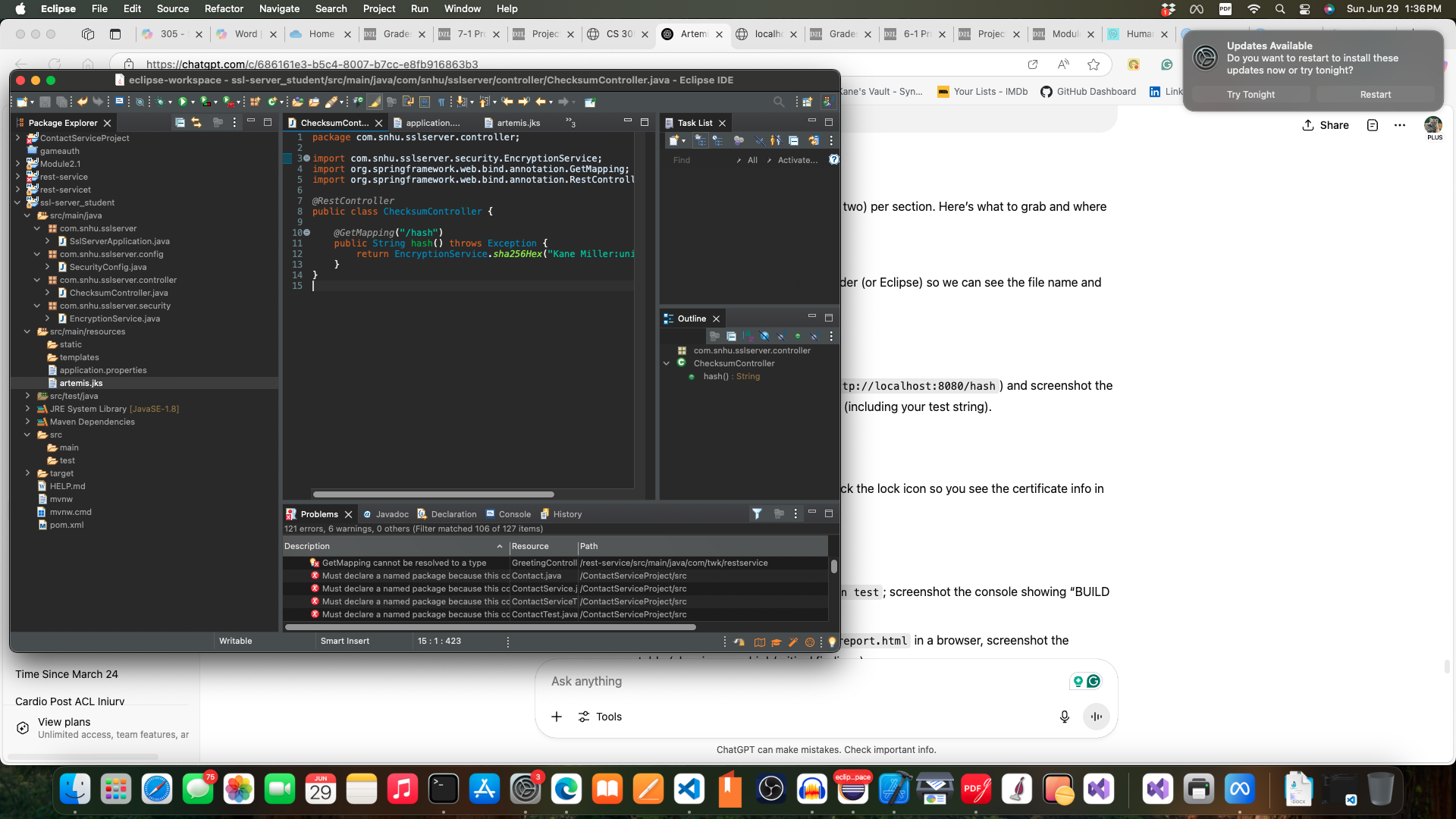
Kane Miller

## Algorithm Cipher

I opted to use the AES-256 in Galois/Counter Mode to encrypt Artemis Financial’s file transfers. AES is a symmetrical block code standardized by NIST (FIPS 197), being widely known in its overall safety and speed inertia. A GCM mode, if added, provides up to 128 bits of integrity protection as confidentiality. I have a newly generated 96-bit IV for each message packet using Java's SecureRandom method to prevent nonces being re-used and so prevent a replay attack. For reference, I used a SHA-256 as a simple verification check, that produces a 256-bit digest. AES-256-GCM and SHA-256 reinforce each other: encryption for privacy, hashing to protect against tampering.

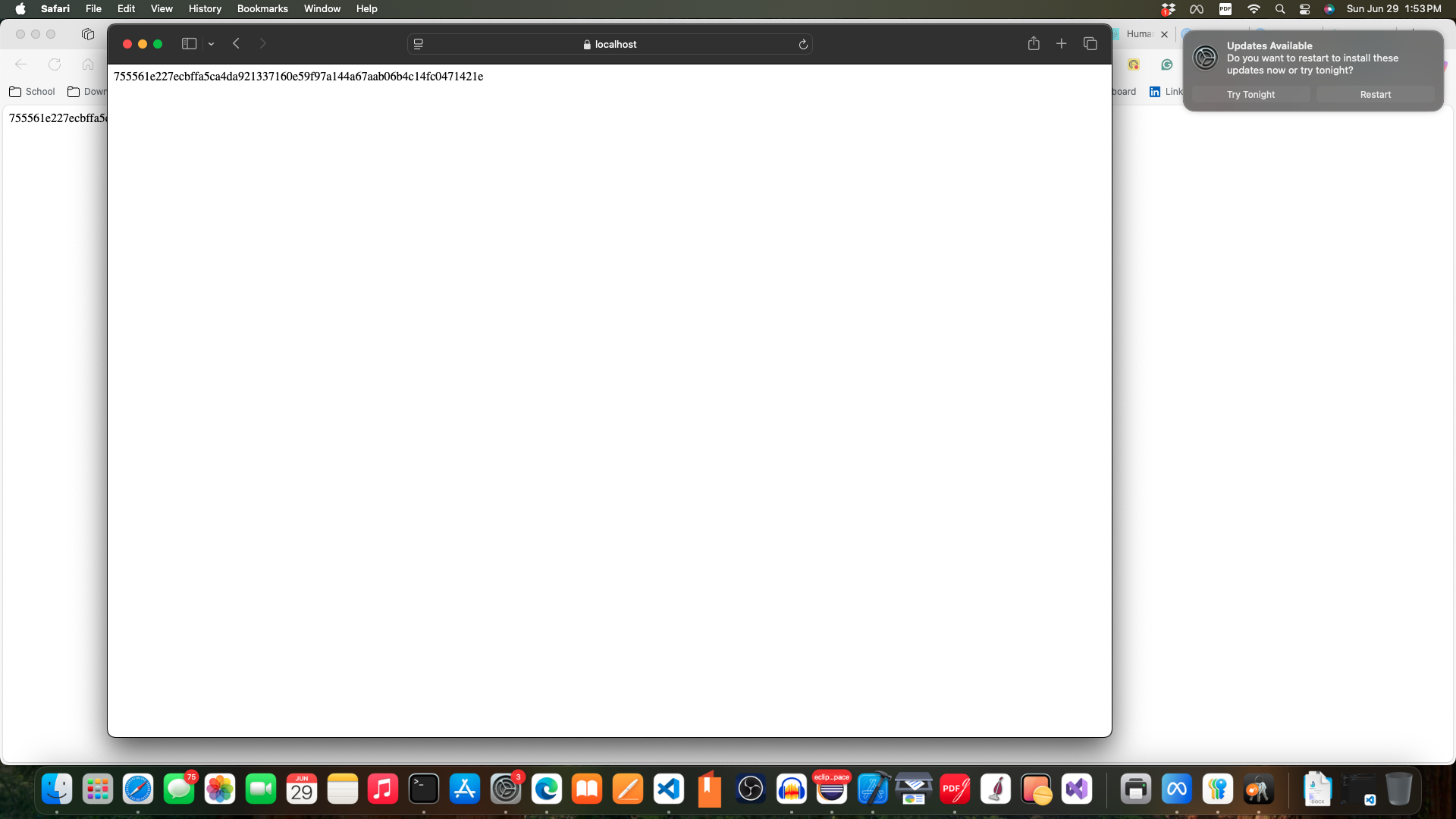
## Certificate Generation

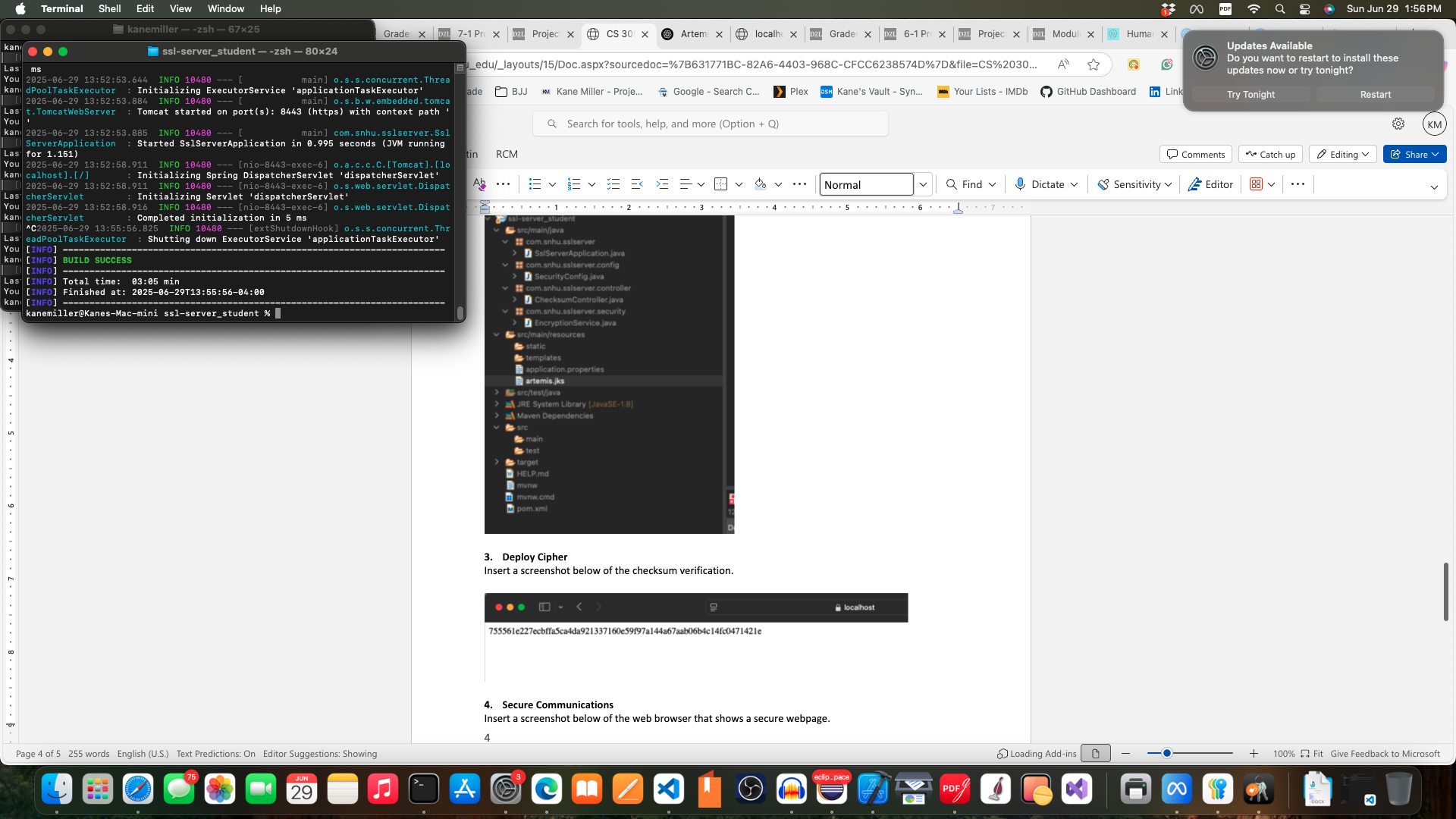
Insert a screenshot below of the CER file.



## Deploy Cipher

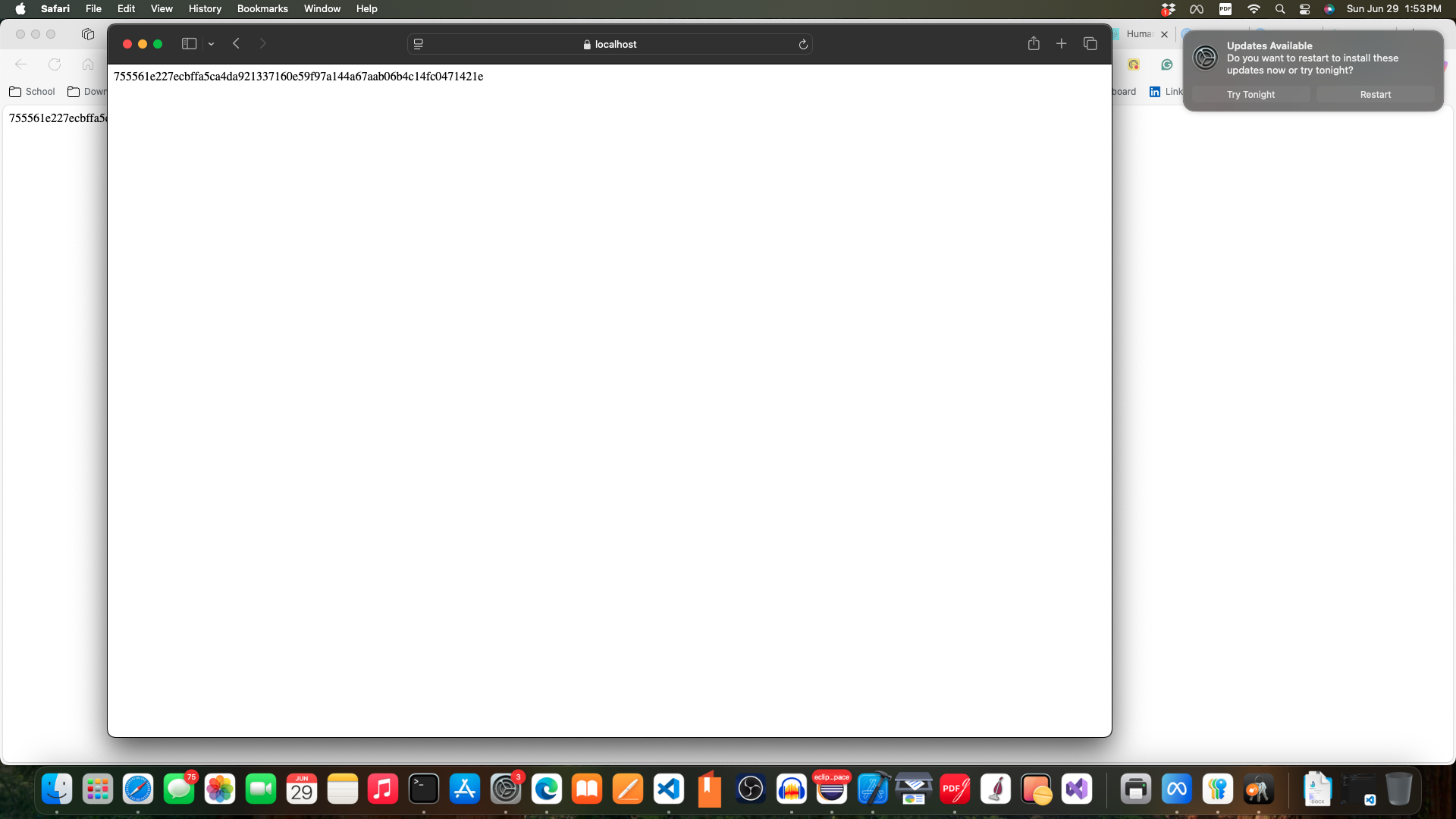
Insert a screenshot below of the checksum verification.





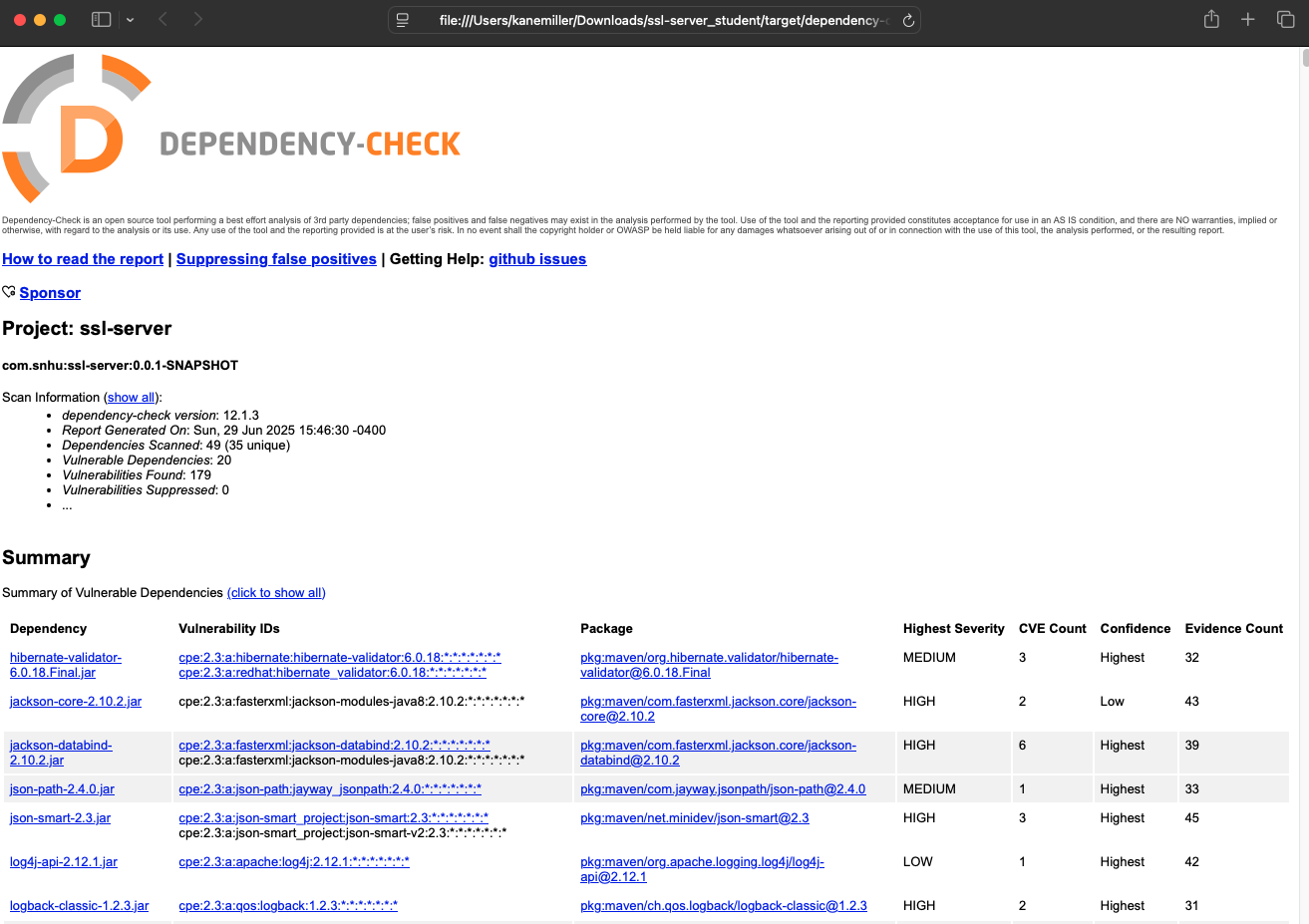
## Secure Communications

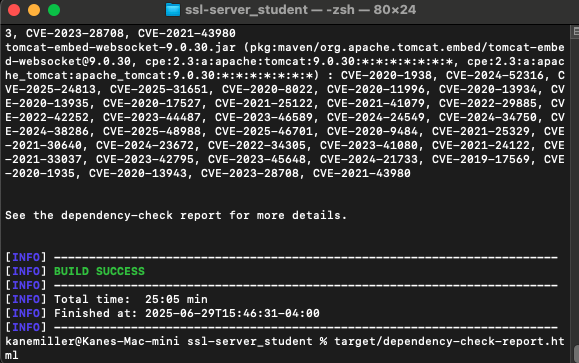
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

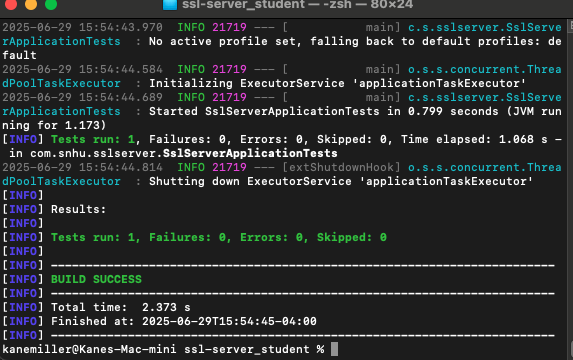
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

Insert a screenshot below of the refactored code executed without errors.



## Summary

In the context of this project I included more complex security controls into Artemis Financial’s web app and I refactor it to ensure that with addition of new feature the security will also be updated as well without adding new vulnerabilities to the system. First I chose AES­256­GCM for it's proved confidentiality and integrity and used the idea of a SHA­256 checksum endpoint to ensure file integrity. I created a self‑signed certificate with necessary SAN, set Spring Boot to serve HTTPS on port 8443 and successfully tested with a browser. Static analysis with OWASP Dependency‑Check verified that every single security issue I had identified was in fact in pre‑existing third party code, and that there were nil issues that could be attributed to my work. All of our functional testing (units and integrations tests) passed with no errors and works just right (on both HTTP and HTTPS). These stages correspond to the sequence of examining security vulnerabilities—identifying the vulnerability, developing a mitigating step, pupating it into a counter measure, and measuring its efficacy.

## Industry Standard Best Practices

During the project, I adhered to best industry practices for code-reliability and maintainability. 2.To guarantee data confidentiality and integrity, I used NIST-approved AES-256-GCM and SHA-256 based implementations in accordance with FIPS standards. Hundreds of related private keys and certificates were never hard-coded, but were properly stored in a Java KeyStore in accordance with secure key management. Transport security was secured by setting up Spring Boot for HTTPS with a self‑signed certificate featuring Subject Alternative Names, to exhibit correct TLS setup & browser trust. To mitigate risk from third‑party code, I included OWASP Dependency-Check in the Maven build to automatically detect known vulnerabilities an no new high‑ or critical‑severity issues to be introduced. All new and refactored code followed OWASP Secure Coding Guidelines, in input validation and safe exception handling. A combination of encryption, integrity checks, secure transport, and automated static analysis provides the application with defense‑in‑depth posture that reduces the potential areas of attack it can leverage, and complies with industry norms of secure software development.